SOCMill

G Code Macro Generation Utility

Technical Reference Manual
V1.15

www.soc-robotics.com

Contents

Introduction

SOCMill (SOCMill.exe) is a free command line (console) application from SOC Robotics, Inc., for PC's running Microsoft Windows™ that translates high level milling and lathe commands written as text based scripts into equivalent G Code. There is no limit to the length of a script and all script commands are printable ASCII. G Code can be mixed with script commands. A line is recognized as a comment if the first character is a semicolon, percent sign, forward slash, opening parenthesis or double quotation mark. SOCgc viewer is an openGL G Code viewer included with SOCMill. SOCgcviewer.exe uses glut32.dll which is included in the project folder.

SOCMill significantly reduces the time it takes to create G Code programs that perform relatively complex operations by using simple high level commands entered as text using a text file editor. Some operations, such as pocket milling, could not be coded by hand or would require an expensive CAD/CAM program.

The resulting G Code can be directly downloaded to a G Code processor such as the GenY32, GenX, GenZ, MK4FQ or MK5FQ controller using another SOC Robotics utility called **`ngload.exe`**. **`ngload.exe`** "drip feeds" one line of G Code at a time to the controller. This prevents the controller from overflowing it's receive buffer.

The resulting G Code is also compatible with 3rd party controllers such as ArtSoft's Mach3, but SOCMill does not yet generate or support some G Codes. Future versions of SOCMill are being enhanced to support these. G Code files generated by SOCMill can be edited with a text editor such as Windows Notepad and special G Codes you want to use with your controller can be added as needed.

Basic drilling and milling operations such as drilling a hole or a more complex operation such as milling a pocket are supported. By combining several script commands it is possible to quickly build up a sophisticated sequence of milling operations. For example, the **`millpath`** command allows the user to specify a sequence of points that are milled repeatedly until the desired depth is reached. Lathe operations such as creating a radius, arc or slope are supported.

Many of the commands support left or right side and inside or outside milling in which the mill bit is moved by half its diameter to one side or the other so the desired dimensions are achieved. A few of the commands support a finishing mill function that allows the user to specify finishing passes. This feature is important when using a light duty milling machine such as the Sherline that has a degree of flex.

Although the program assumes dimensions are specified in inches they can just as well be mm. Units of measure used in SOCMill will be interpreted according to the selection of "native units" in your controller application or hardware. SOCMill generates a sequence of G-codes to drive the x, y, z and a axes by executing milling instructions read from a source script file. All milling operations are with respect to a starting position. The start position is the center tip location of the tool bit as the script begins execution. All 4 axes x, y, z, and a are set to zero (0.0) at this start position. Wherever the tool happens to be when script execution begins, this point is set to zero on all axes.

The program is continually being upgraded with new features. We use the program extensively in-house for our milling operations so expect to see it evolve over time.

An extensive set of example scripts are included with the SOCMill distribution. Note that this is a free application and as such does not come with any support. Use of the program is at your own risk.

Using SOCMill

The following points explain how to use `SOCMill`.

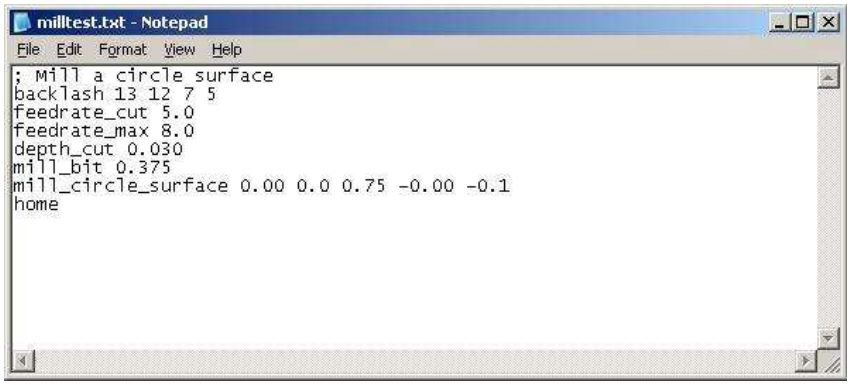
1) Open Notepad or any other word processing application that can save files as simple text (.txt) and type Global Parameters and Commands, each on a separate line. Use a semicolon on a line by itself

to create space between blocks of commands for a more readable file, or with additional text to create non-executable comments. Other characters that can begin a comment line are: %, /, (, and ". Upper case characters can be used in scripts, but `SOCMill` converts all text to lower case characters, therefore comments, commands and parameters will not contain capital letters in the generated G Code output file.

2) Save your script file in the folder containing the `SOCMill.exe` application with your choice of name, such as "bearing.txt".
3) Launch `SOCMill.exe` in the usual way apps are started in Windows (by selecting it and pressing the Enter key on your keyboard, by double-clicking on it with the left mouse button, or by going to the Start Menu and using the "Run" option).
4) When the `SOCMill` console window appears, it will prompt you for an input file name. At the prompt, type the name of your script file (example: bearing.txt) and press the Enter key.
5) **SOCMill** will now prompt you for an output file name. This can be anything you like, except the original file name of the script input file (examples: bearing.gc or brg_1.out). Type the output file name and press the Enter key. (If the input and output filenames are the same, **SOCMill** console window will display: "Error: Input and output files have the same name "bearing.txt". Press any key to continue.")
6) If the input file name does not exist in the **SOCMill.exe** working directory, **SOCMill** will report an error. (example: "Can't open input file bearing.txt – Press any key to continue". Otherwise, **SOCMill** will generate the G Code output file in the **SOCMill** working directory.
7) Before closing **SOCMill**, you may want to inspect the console window printout and ensure no errors occurred because of missing parameters or spelling errors. To hold this window on screen, use the **`"wait"`** Global Parameter in your script. All script command parameters (as defined in the section **Detailed Command Description**) must appear with that command, even if they have not changed or are equal to zero.
8) The output file containing your G Code program is now ready to use with your software or controller. If you are using an SOC Robotics product with an on board G Code processor, use **`ngload.exe`** (available at our download page of applicable products) to feed the G Code file to your device. If you are using a 3rd party controller or software such as ArtSoft's Mach3, load and run your G Code file in the usual way.

EXAMPLES

First create a script text file, like the example **`milltest.txt`** shown in the screenshot `below.`



In this example the backlash numbers are specific and unique to SOC Robotics, Inc. G Code controllers (see the GenY32 Technical Manual).

Commands used in the above example:

`feedrate_cut` is the speed at which the mill bit moves through the material parallel to the active plane.
`feedrate_max` is the speed the bit moves from one mill point to the next.

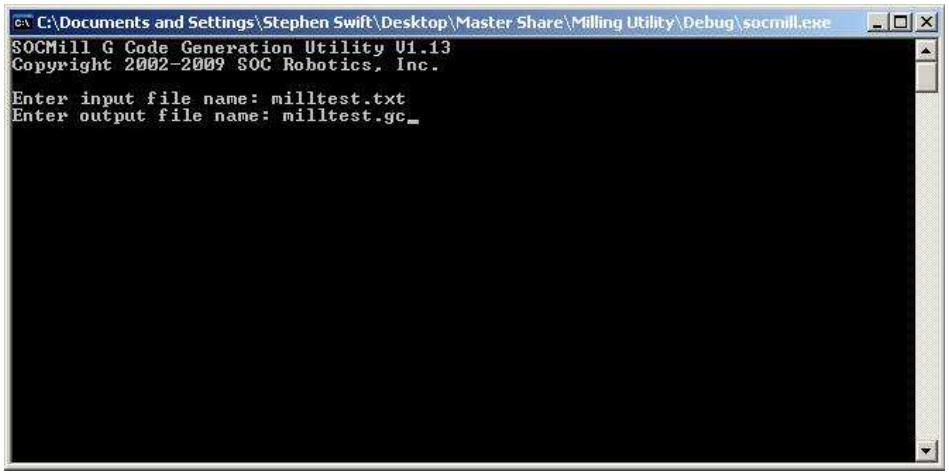
depth_cut is the depth of each milling cut pass– this depends on the mill, speed and stiffness of the machine - 0.030 in is typical for a Sherline mill and a 0.375in bit.

mill_bit sets the diameter of the milling tool.

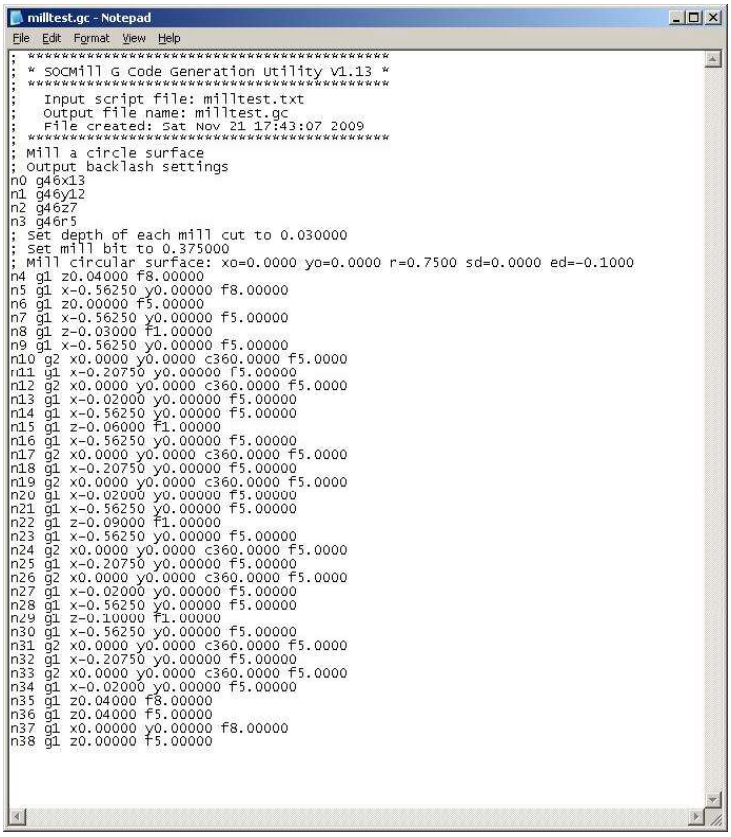
mill_circle_surface is a high level milling command that removes all material in a circle centered at (0, 0), radius 0.75 in, from a depth starting at 0.0 and ending at –0.1 – essentially this is a circular pocket milling operation. Note that **SOCMill** automatically moves the tool to the correct starting position so that the correct circle radius is achieved.

The mill bit is assumed to be at the origin although this is not necessary. If a second circular pocket milling operation is required to a different depth it can be entered on the next line.

Start the program by typing **SOCMill.exe** (SOCMill.exe <CR>) on the command line and enter the input text file name. **SOCMill** will then prompt you for an output file name. Type the output file name and press **Enter**.



This creates the output file milltest.gc containing the following G Code:



ngload.exe can now be used to send the file `milltest.gc` to a G-code controller through a serial port.

ANOTHER EXAMPLE:

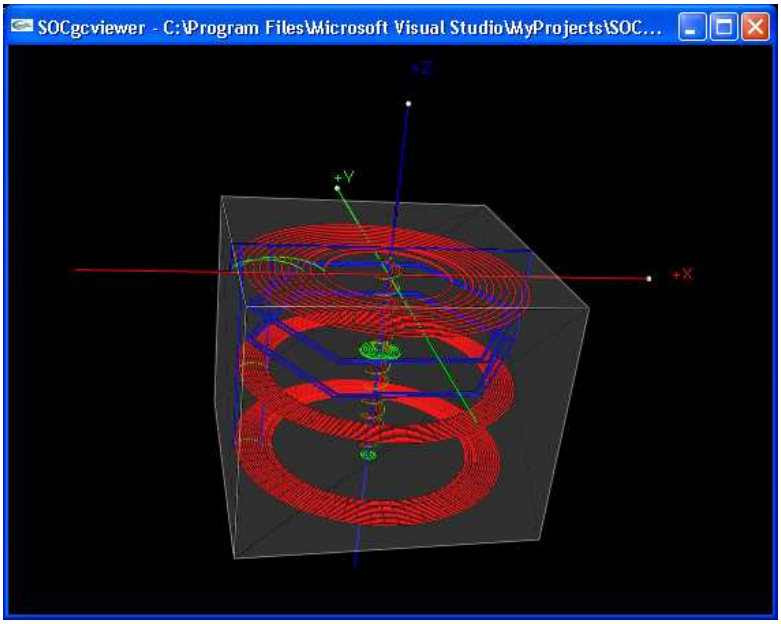
Shown below is another example script file written in Windows ™ Notepad. This one illustrates a few more commands and some of the Global Parameters that can be used at any point in the script.

```
verbose y
arc_format c
re_wind y

cut_rough .010
cut_finish .0025
lift_height 0.100
feedrate_rough 7.0
feedrate_finish 8.0

change drill_bit 4
feedrate_plunge 1.0
peck_drill n 0.0 0.0 .025 .025 0.0 -0.200

change drill_bit .246
feedrate_plunge 3.0
peck_drill n 0.0 0.0 .125 .025 0.0 -0.375

change mill_bit .2485
feedrate_rough 5.0
mill_pocket .400 w w 0.0 -0.300

cut_rough .050
mill_cylinder .750 0.350 0.0 0.0

cut_rough .030
feedrate_rough 3.0
mill_hex .700 .5625 0.0 -0.125
mill_hex .700 .5625 0.0 -0.250

change drill_bit 4
feedrate_plunge 1.0
peck_drill n 0.0 0.0 .025 .025 0.0 -0.200

change drill_bit .246
feedrate_plunge 3.0
peck_drill y 0.0 0.0 .125 .025 0.0 -0.800

cut_rough .050
change mill_bit .2485
feedrate_rough 7.0
mill_cylinder .700 0.200 0.0 0.0

cut_rough .010
feedrate_rough 2.0
mill_cylinder .760 .560 0.0 -0.375
mill_cylinder .760 .560 0.0 -0.700
feedrate_rough 5.0
mill_cylinder .560 .499 0.0 -0.700
feedrate_rough 7.0
mill_pocket .307 w w 0.0 -0.750

cut_rough .015
mill_mode i
climb n
change mill_bit .290
feedrate_rough 7.0
mill_thread .365 .297 16 0.0 -.7

climb y
feedrate_rough 10.0
mill_thread .375 .297 16 0.0 -.7
```

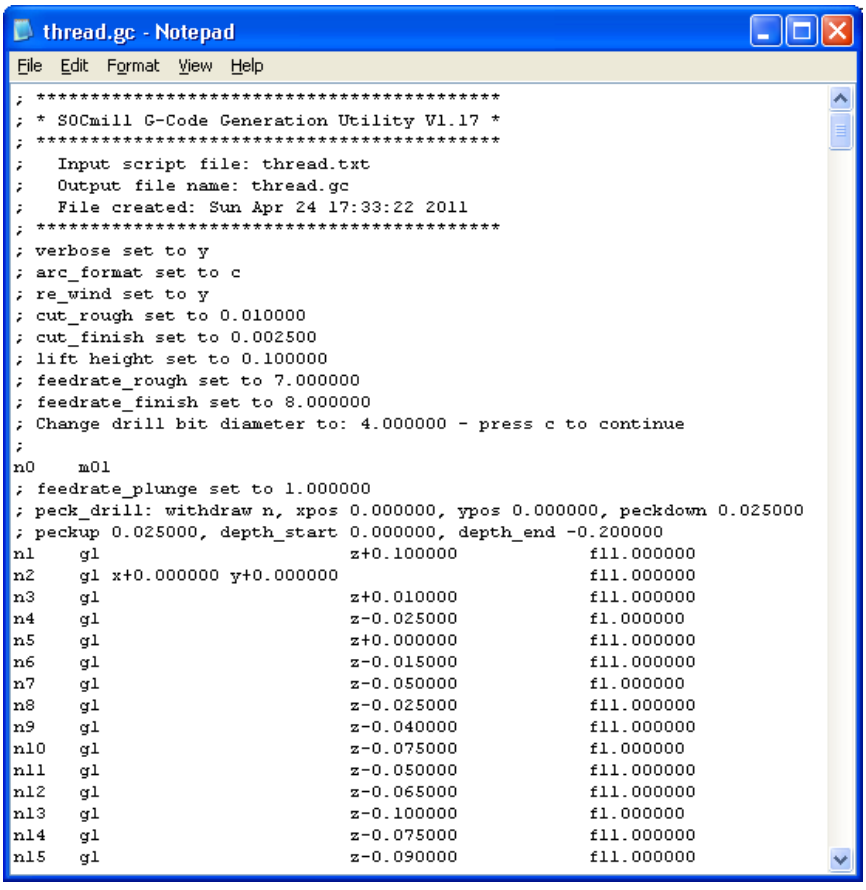


```
wait n
toolpath
```

The last Global Parameter, "toolpath", starts a second application called "SOCgcViewer"" that is a 3D OpenGL display window showing the path of the tool tip during all the operations in the generated g-code output file. The toolpath for the 'thread.txt" example is shown below.



The script generates an output file with almost 500 numbered lines of g-code. The first part of this output file is shown below:



Global Parameters Summary

Default values for Global Parameters are loaded into SOCMill at program startup.

(Note: At this time, SOCMill does not store user changes to Global Parameters. We are updating SOCMill and adding this feature very soon – Note added Sept. 09, 2010)

Global Parameters select various milling modes for and set parameters common to most of the canned operations found in the Command Summary. Global Parameters can be changed as often as needed during a script, to achieve the desired behaviour for each Command. A Global Parameter remains in effect for all applicable commands for the duration of the script, until altered by another call to that same parameter in the script.

Parameter	Options	Example	Description
lift		lift	lift tool between operations
no_lift		no_lift	do not lift tool between operations
mill_dir	c w	mill_dir c	select cw (c) or ccw (w) for arc or circles
mill_mode	c i o	mill_mode o	select center, inside, outside milling (tool rad compensation)
ij_mode	a i	ij_mode i	select absolute or incremental ij_mode for center-format arcs
distance_mode	a i	distance_mode a	select absolute or incremental distance_mode
climb	y n	climb y	select climb or standard milling
thread_hand	r l	thread_hand r	select right or left-hand threading
numfinishcuts	integer >= 0	numfinishcuts 1	set the number of finish cuts
repetitions	integer >= 0	repetitions 2	set the number of times to repeat the last finish cut
cut_rough	decimal > 0	cut_rough .025	set the width of rough cuts in milling operations
cut_finish	decimal > 0	cut_finish .004	set the width of finish cuts in milling operations
depth_cut	decimal > 0	depth_cut .0625	set the depth of plunge steps in milling operations
lathe_cut	decimal > 0	lathe_cut 0.010	sets the depth of cuts in lathe operations
mill_bit	decimal > 0	mill_bit 0.375	set the diameter of the current tool
drilldiam	decimal > 0	drilldiam .4375	set the diameter of the current drill
feedrate_cut	decimal > 0	feedrate_cut 5.0	set feedrate for operations without finish cuts
feedrate_rough	decimal > 0	feedrate_rough 11.0	set feedrate for operations with roughing cuts
feedrate_finish	decimal > 0	feedrate_finish 7	set feedrate for operations with finishing cuts
feedrate_plunge	decimal > 0	feedrate_plunge 1.75	set feedrate for mill or drill plunge (z-)
feedrate_max	decimal > 0	feedrate_max 25	set feedrate for safe_z and g0 traverse moves
feedrate_lathe	decimal > 0	feedrate_lathe 7.5	set feedrate for lathe operations
off_contact	decimal > 0	off_contact .007	set the clearance for peck_drill plunge
lift_height	decimal > 0	lift_height 0.100	set the z position for tool height between operations
verbose	y n	verbose n	select formatted g-code output (y) or non-formatted (n)
arc_format	r c	arc_format c	set the default format for arc moves (radius or center)
re_wind	y n	re_wind n	adds "M30" rewind command to end of g-code output

*** Not yet implemented ***

work_z	decimal > 0	safe_z 0.250	moves the z axis to a working height very near the material
safe_z	decimal > 0	safe_z 0.250	moves the z axis to a height that clears all fixtures
tool_z	decimal > 0	tool_z 0.250	moves the z axis to a height that allows tool change

Command Summary

The following commands are supported, each with a detailed description in the next section:

change home mill_bit - Select a new mill bit - returns to home, waits for tool change
change nohome drill_bit - selects a new drill bit - returns to home, waits for tool change
rotate - Rotate number of degrees + or -
backlash – Set backlash for each axis - same as G Code commands
drill_hole - Drill hole at (x,y) from depth_start to depth_end
peck_drill – Drill hole using pecking motion at (x,y) to depth d
drill_array – Drill an array of holes given separation of dx,dy and length x,y
drill_circle – Drill an array of holes in a circle of radius

mill_hole - Mill a hole at (x, y) to depth
mill_line - Mill a straight line between two end points - centered or left/right of center
mill_arc - Mill inside/outside/center arc given starting point, origin and angle degrees
mill_circle - Mill circle inside/outside/center with center at x y, radius and depth
mill_circle_inside - Mill outside radius
mill_circle_outside - Mill inside radius
mill_circle_surface - Mill a pocket circle
mill_square – Mill a square given center, x y length, start end depth
mill_surface - Mill surface given center, x y length, start end depth
mill_surface_square - Mill surface given center, x y length, start end depth square corners
mill_box - Mill inside/outside/center of box given center, x length and y length at depth
mill_round_box - mill inside/outside/center of box with round corners
mill_path - Mill a sequence of points - closed or not closed
mill_pocket – mill an inside circular pocket
mill_thread – mill an inside or outside thread, right-hand or left-hand
mill_cylinder – mill an outside cylinder
mill_sprocket – mill a straight-toothed pick and place feeder sprocket (proprietary)
mill_hex – mill an outside hexagon

lathe_diameter - Lathe length of round bar to final depth given xstart and xend
lathe_slope - Lathe taper given xstart, xend, ystart and yend position
lathe_angle - Lathe taper given xstart, xend and angle
lathe_radius – Lathe round on end of rod
lathe_pulley - Cut pulley on lathe given diam,edge,width,depth,depthwidth,enddiam

align - Align the x, y and z axis - use left front corner
home - return to home position (ie xinitial, yinitial, zinitial)
calibrate - compute backlash for x, y and z
home_shift - set new home position xinitial, yinitial, zinitial, ainitial
home_set - set new home position in MC433G
home_zero - zero xinitial,yinitial,zinitial at current position
wait – do not exit program until key is pressed

SOCviewer – Can only appear as the last Command in a script file – invokes an OpenGL toolpath display

Special Functions:

disk_array – mill an array of disks of diameter d

drill_encoder_disk – Drill circular set of holes

mill_encoder_disk - mill encoder disk - radius, center hole radius, no teeth, tooth width, tooth height, depth

Certain operations cannot be performed if the tool is too large, such as milling the inside of a box or circular hole that is smaller than the mill bit. For most situations in which this would be an issue, SOCMill checks parameters, compares this to the current tool diameter and generates an error message in the SOCMill console window printout.

Cartesian Coordinate system: The x-axis is left/right increasing to the right. The y-axis is in/out increasing to the back. The z-axis is up/down positive up. The a-axis is counterclockwise plus degrees and clockwise negative degrees.

xinitial, yinitial, zinitial, ainitial is the zero point - all movements are with respect to this point - in this way the initial point can be moved to duplicate a complex operation

xstart, ystart is the front left corner of the piece.

Tool changes are accommodated by writing a colon in the first column position of a new line - type 'c' to continue - what about backlash?

Detailed Command Description

change mill_bit

Select a new mill bit – generate pause command and wait for tool change.

Syntax: **change mill_bit** tooldiam

```
Example:   change mill_bit  0.250
```

change drill_bit

Select a new drill bit - generate pause command and wait for tool change

Syntax: **change drill_bit** drilldiam

```
Example:   change drill_bit  .094
```

drill_hole

Drill a hole at location (**xpos**, **ypos**) from **depth_start** to **depth_end** continuously at **feedrate_plunge** without chip-breaking.

Syntax: **drill_hole** xpos ypos depth_start depth_end

```
Example:   drill_hole  1.250 2.250 0.000 -0.3125
```

peck_drill

Peck-drill a hole at location (**xpos**, **ypos**) from **depth_start** to **depth_end** in steps at **feedrate_plunge** with chip-breaking. Amount of downward plunge is set by the **peckdown** parameter. Type of chip-breaking is selected by the withdraw parameter (**y** for yes, **n** for no). If **y**, the drill completely withdraws up to **depth_start**. If **n,** the drill withdraws up by the amount of the **peckup** parameter. After withdraw, the drill returns at **feedrate_max** to the current depth plus the amount of the **off_contact** Global Parameter, then continues drilling at **feedrate_plunge**. This action repeats until **depth_end** is reached.

Syntax: **peck_drill** withdraw xpos ypos peckdown peckup depth_start depth_end

```
Example:   peck_drill  y 1.250 2.250 0.030 0.100 0.000 -0.3125
```

mill_line

Mill a straight line from (**xstart**, **ystart**) to (**xend**, **yend**) and from **depth_start** to **depth_end** at **feedrate_cut**. The **mode** parameter (**l** for left, **r** for right, **c** for center) sets tool radius compensation. If mode is set to **c**, no tool radius compensation is used.

Syntax: **mill_line** mode xstart ystart xend yend depth_start depth_end

```
Example:   mill_line  c 0.0 0.5 1.125 0.875 0.0 -0.500
```

mill_arc

Mill inside/outside of an arc given starting point, origin and angle in degrees.

Syntax: **mill_arc** mode xpos ypos xorigin yorigin angle depth_start depth_end


```
Example:   mill_arc  c 1.25 2.25 2.0 3.0 90 0.0 -0.025
```

mill_circle

Mill circle with center at origin, radius and depth.

Syntax: **mill_circle** xorigin yorigin radius depth_start depth_end

```
Example:   mill_circle  1.25 2.25 0.45 0.0 -0.025
```

mill_circle_helix

Mill circle of given **diameter** with center at (**xorigin**,**yorigin**) from **depth_start** to **depth_end** using a helical plunge.

Syntax: **mill_circle_helix** xorigin yorigin diameter depth_start depth_end

```
Example:   mill_circle_helix  1.25 0.75 0.45 0.0 -0.025
```

mill_circle_inside

Mill inside radius.

Syntax: **mill_circle_inside** xorigin yorigin radius depth_start depth_end

```
Example:   mill_circle_inside  1.25 1.00 0.45 0.0 -0.025
```

mill_circle_outside

Mill outside radius.

Syntax: **mill_circle_outside** xorigin yorigin radius depth_start depth_end

```
Example:   mill_circle_outside  1.25 1.75 0.45 0.0 -0.025
```

mill_circle_surface

Mill a pocket circle from **depth_start** to **depth_end.**

Syntax: **mill_circle_surface** xorigin yorigin radius depth_start depth_end

```
Example:   mill_circle_surface  .35 .7 .500 0.0 -.250
```

mill_surface

Mill a surface given by center, x length and y length from **depth_start** to **depth_end.**

Syntax: **mill_surface** xcenter ycenter xlen ylen depth_start depth_end

```
Example:   mill_surface  2.0 2.2 4.0 3.0 0.0 -0.065
```

mill_surface_square

Mill a surface given center, x length and y length from **depth_start** to **depth_end.**

Syntax: **mill_surface_square** xcenter ycenter xlen ylen depth_start depth_end

```
Example:   mill_surface_square  2.0 1.2 4.0 3.0 0.0 -0.065
```

mill_square

Mill a box from **depth_start** to **depth_end.**

Syntax: **mill_square** xcenter ycenter xlen ylen depth_start depth_end

```
Example:   mill_square  0.0 -0.5 0.500 0.000 0.0 -1.000
```

---

mill_box

Mill inside/outside of box given center, x length and y length, from **depth_start** to **depth_end.**

Syntax: **mill_box** mode xcenter ycenter xlen ylen depth_start depth_end

```
Example:   mill_box  i 0.0 -0.5 0.500 0.000 0.0 -1.000
```

---

mill_round_box

Mill inside/outside of box with round corners.

Syntax: **mill_round_box** mode xcenter ycenter xlen ylen corner_radius depth_start depth_end

```
Example:   mill_round_box  o 1.312 0.812 2.0 1.25 .312 0.000 -0.75
```

---

mill_path – (see detailed description at the end of this Command Summary)

Mill an inside, center, or outside path defined by point coordinates in a text file.

Syntax: **mill_path** depth_start depth_end filename

```
Example:   mill_path  0.0 .250 path.txt
```

---

mill_pocket

Mill circular inside pocket.

Syntax: **mill_pocket** PocketDia sDirRough sDirFinish depth_start depth_end

```
Example:   mill_pocket  1.625 c w 0.0 -.063
```

---

mill_thread

Mill inside or outside thread, right or left hand, climb or standard milling.

Syntax: **mill_thread** major_diam minor_diam tpi depth_start depth_end

```
Example:   mill_thread  0.375 0.297 16 0.0 -.700
```

---

mill_cylinder

Mill outside cylinder ***(only uses climb milling and location is (0,0), will update shortly!) ***

Syntax: **mill_cylinder** stock_diameter finish_diameter depth_start depth_end

```
Example:   mill_cylinder  1.010 0.9375 .015 –0.625
```

---

mill_sprocket (proprietary)

Mill a pick and place feeder sprocket with center hole, number of teeth, tooth width and height, to depth.

Syntax: **mill_sprocket** diskdiam hole_diam num_teeth tooth_width tooth_height depth_start depth_end

```
Example:   mill_sprocket  1.875 0.250 56 .075 .055 0.0 -0.125
```

---

mill_hex

Mill an outside hexagon.

Syntax: **mill_hex** stock_diameter across_flats depth_start depth_end

```
Example:   mill_hex  0.752 .5625 0.0 -.375
```

---

mill_encoder_disk (proprietary)

Mill an encoder disk.

Syntax: **mill_encoder_disk** diskdiam hole_diam no_teeth tooth_width tooth_height start_height end_height

```
Example:   mill_encoder_disk  0.875 0.250 56 .075 .055 0.125 0.150
```

---

lathe_diameter

Cut diameter on lathe.

Syntax: **lathe_diameter** xstart ystart xend yend

```
Example:   lathe_diameter 2.125 0.0 .25 .05
```

---

lathe_slope

Cut bevel using point-slope on lathe.

Syntax: **lathe_slope** xstart ystart xend yend

```
Example:   lathe_slope 1.000 0.0 .25 .05
```

---

lathe_angle

Cut bevel using point-angle on lathe.

Syntax: **lathe_angle** xstart ystart yend angle

```
Example:   lathe_angle 2.125 0.0 .25 45
```

---

lathe_radius

Cut radius on lathe.

Syntax: **lathe_radius** xstart ystart radius

```
Example:   lathe_radius 2.125 0.0 .250
```

---

lathe_pulley (proprietary)

Cut pulley on lathe.

Syntax: **mill_pulley** xstart ystart diam edgelength slotwidth slotdepth slotwidthatdepth stubwidth stubdepth

```
Example:   mill_pulley 2.125 0.0 1.125 .05 0.625 2.125 0.0 1.125 .05
```

---

drill_encoder_disk (proprietary)

Drill encoder disk - diameter, center hole diameter, number of teeth, start height, end height.

Syntax: **drill_encoder_disk** disk_diam hole_diam no_teeth start_height end_height


```
Example:   drill_encoder_disk  0.875 0.250 56 .075 .055
```

rotate

Rotate number of degrees + or – at **feedrate_cut.**

Syntax: **rotate** degrees

```
Example:   rotate -45.00
```

backlash

Same as G Code commands.

Syntax: **backlash** ix iy iz ia

```
Example:   backlash .0015 .002 .004 0.0
```

align

Align the x, y and z axis - use left front corner.

Syntax: **align**

```
Example:   align
```

disk_array (proprietary)

Drill encoder disk.

Syntax: **disk_array** disk_diam

```
Example:   disk_array  0.875
```

drill_array (proprietary)

Drill encoder disk.

Syntax: **drill_array** xpos ypos width height xstepincrement ystepincrement depth_start depth_end

```
Example:   drill_array  .25 .5 2.0 1.0 .125 .125 0.0 -.175
```

home

Return to home position (ie xinitial, yinitial, zinitial).

Syntax: **home**

```
Example:   home
```

calibrate (***not yet implemented***)

Compute backlash for x, y and z.

Syntax: **calibrate** ix iy iz ia

```
Example:   calibrate .0015 .002 .004 0.0
```

home_set

Set new home coordinates xinitial, yinitial, zinitial, ainitial.

Syntax: **home_set** xnewpos ynewpos znewpos anewpos


```
Example:   home_set 2.5 0.0 -0.125 0.0
```

home_zero

Zero xinitial, yinitial, zinitial ainitial at current position.

Syntax: **home_zero**

```
Example:   home_zero
```

toolpath

Use to invoke **SOCgcviewer** to display an interactive 3D graphic of the toolpath after script finished.

Syntax: **toolpath**

```
Example:   toolpath
```

wait

Use to review notes and warnings generated by **SOCMill**. Press any key to close the **SOCMill** console.

Syntax: **wait**

```
Example:   wait y
```

mill_path - A Detailed Description

mill_path is a command that takes three parameters, **depth_start**, **depth_end** and the name of a file containing the desired path. This separate path file is a simple text file containing coordinate pairs (x,y) that define the end points of line segments along the path. Path files can also contain single letter sub-commands that define arc moves of both radius-format (**r**) and/or center-format (**c**) arcs. A short SOCMill script file using **mill_path** might look like the example shown below. Although this short script was easy to write and only took a few minutes, it generates about 50 or more lines of g-code to mill the path, depending on the number of points in the path. The maximum number of path points allowed is 10,000.

```
mill_mode o
climb y
ij_mode a
lift_height .300
feedrate_max 20
feedrate_cut 15
depth_cut 0.1
mill_path  0.0  -.55  path.txt
```

This script will mill a path given by a file named (in this case) path.txt. An example of a path file used to clockwise-mill an outside square-cornered rectangle is shown below. A simple text file contains the (x,y) locations as coordinate pairs and should be in the same directory as the script file and SOCMill.exe.

```
0.0    0.0
0.0    1.0
2.0    1.0
2.0    0.0
0.0    0.0
```


The rectangle path shown above used with the previous script example will mill the path repeatedly at step depths defined by Global Parameter **depth_cut** until **depth_end** is reached. **mill_path** can mill both closed and open paths. The start and end points are the same for a closed path, for an open path they are different. If the first and last point pair in the path file are different (open path), the tool will be raised to safe_z at the end of the path and then will move back to the starting point, plunge to the new step depth at **feedrate_plunge** and continue milling the path at the new step depth.

The next example shows the same rectangle, this time with 0.25" radius round corners, defined with the "**r**" sub-command. The point pair just before the line with the "**r**" is the start point of the radius (and also the end point of the previous straight line) and the point pair just after it is the end point of the radius (and the start point of the next straight line or arc). The value for the length of the arc radius is on the same line as the r sub-command. (Notice that the start and end points of the path are the same (closed path). Because of this, the tool will not be raised to safe_z, when it reaches the end point. It will plunge at **feedrate_plunge** directly to the next step depth and continue milling the path.)

```
0.0     0.25
0.0     0.75
r       0.25
0.25    1.00
1.75    1.0
r       0.25
2.0     0.75
2.0     0.25
r       0.25
1.75    0.0
0.25    0.0
r       0.25
0.0     0.25
```

Arcs defined with the "**r**" sub-command are limited to 180 degrees or less, because only the radius is given. This is not enough information for the program to logically determine which of the four arc possibilities to use. Radius-format arcs are therefore limited to the two choices allowed by arcs of 180 degrees or less. This is simply a mathematical reality, not a limitation of the SOCMill.exe application.

However, we have the "**c**" sub-command for center-format arcs, which allows us to define arcs up to 360 degrees. This is possible because the location of the center of the arc is given with the command and the radius is instead calculated by the program, so we have all the information we need to define any circular arc, as shown in the path file below.

```
0.0     0.25
0.0     0.75
c       0.25    0.75
0.25    1.00
1.75    1.00
c       1.75    0.75
2.00    0.75
2.00    0.25
c       1.75    0.25
1.75    0.00
0.25    0.00
c       0.25    0.25
0.00    0.25
```


The path above defines exactly the same round-cornered rectangle as the previous path using the "**r**" sub-command, except that there are now two values given with the "**c**" sub-command. These are the **i** and **j** values that are the location in absolute coordinates of the center of the arc. We use absolute coordinates for the arc center (in this case) because we defined **ij_mode** as absolute (**a**) in the **mill_path** script file above. We could have defined **ij_mode** as incremental (**i**) and the path file would now look like the example shown below.

```
0.0     0.25
0.0     0.75
c       0.25    0.0
0.25    1.00
1.75    1.00
c       0.00    -.25
2.00    0.75
2.00    0.25
c       -.25    0.0
1.75    0.00
0.25    0.00
c       0.00    0.25
0.00    0.25
```

A question that might come to mind would be "How does SOCMill know when to use a clockwise or a counterclockwise arc?" SOCMill has some intelligent decision making built in, to make most path files as simple as possible. For outside corner turns, SOCMill automatically generates outside (convex) arcs. For inside corner turns, SOCMill automatically generates inside (concave) arcs, since this will be true in most cases, for most paths. However, we have a way to force arcs to ignore the default arc direction. We can use the inside (**i**) and outside (**o**) arc sub-commands. These commands are placed immediately after the "**r**" or "**c**" sub-commands as shown in the example below.

```
0.0     0.25
0.0     0.75
r       i       0.25
0.25    1.00
1.75    1.0
r       i       0.25
2.0     0.75
2.0     0.25
r       o       0.25
1.75    0.0
0.25    0.0
r       o       0.25
0.0     0.25
```

Now the top left and top right arcs will be concave (inside) round corners and the bottom right and bottom left corners will be convex (outside) round corners. The bottom two would have been outside corners by default anyways, because we defined **mill_mode** as outside (**o**) in the **mill_path** script file.

SOCMill performs an analysis of your path file to determine if there are any logical conflicts and will warn you of such a conflict. For instance, if you have defined a path that generally moves in a clockwise direction around the part and you have set **mill_mode** as "**o**" (outside) and **climb** to "**n**" (standard milling), SOCMill will report a conflict, since this is clearly impossible for a right-handed end-mill. (SOCMill assumes that the tools being used have right-handed flutes and turn clockwise in the spindle as viewed from above.) In this case SOCMill will recommend that you either change to climb = "**y**" or will recommend that you re-write

your path file to reverse your path. SOCMill will allow you to run the file even though a conflict is detected, if that is what you need to do.

For arcs involving no turn (straight ahead), SOCMill defaults to convex arcs based on the **mill_mode** and **climb** Global Parameters and also on the general milling direction defined by the path itself (clockwise or counterclockwise). Convex is chosen as the default in straight-ahead moves because of the old machine-shop saying "You can always remove material, but it cannot be put back on, once removed." In other words, SOCMill defaults are designed to err on the side of safety, to reduce the chance of lost time and material. Once again, these defaults can be over-ridden by the "**i**" and "**o**" sub-commands for circular arcs.

Inspection of the resulting path in a 3D tool path display, such as the one built into ArtSoft's Mach3, is very helpful in determining where and when to use these sub-commands, since one can instantly recognize visually when an arc is not as intended. To use SOCMill's built-in 3D viewer, use the **toolpath** Command as the last entry in your script.